Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 306 (2003) 471–484

Theoretical
Computer Science

www.elsevier.com/locate/tcs

RNA secondary structure comparison: exact analysis of the Zhang–Shasha tree edit algorithm

Serge Dulucq*, Laurent Tichit

LaBRI, Université Bordeaux I, 351 cours de la Liberation CNRS U.R.a. 1304, Talence 33405 cedex, France

Received 31 March 2003; accepted 14 May 2003
Communicated by A. Del Lungo

Abstract

We are interested in RNA secondary structure comparison, using an approach which consists to represent these structures by labeled ordered trees. Following the problem considered, this tree representation can be rough (considering only the structural patterns), or refined until an exact coding of the structure is obtained. After some preliminary definitions and the description of the Zhang–Shasha (SIAM J. Comput. 18 (6) (1989) 1245) tree edit algorithm, which is on the one hand the reference when dealing with ordered labeled trees comparison, and on the other hand the starting point of our work, this article will present an exact analysis of its complexity. The purpose of this work is also to lead us to a better comprehension of the parameters of this algorithm, in order to be able to modify it more easily without changing its time complexity to take into account biological constraints that occur when comparing RNA secondary structures.

Keywords: RNA secondary structure; Labeled ordered tree; Complexity analysis; Generating function

1. Introduction

When we want to study the similarity between two RNA molecules, it often becomes interesting to compare not only their sequences, but their structures as well. As two different sequences can produce similar secondary structures, comparisons between

* Corresponding author.
E-mail addresses: dulucq@labri.u-bordeaux.fr (S. Dulucq), tichit@labri.u-bordeaux.fr (L. Tichit).

0304-3975/03/$ - see front matter © 2003 Published by Elsevier B.V.
doi:10.1016/S0304-3975(03)00323-2

secondary structures are necessary to lead to a better understanding of the functions of different RNA molecules. This secondary structure comparison can hence allow to predict with a better efficiency the spatial folding of a given molecule, to help the identification of structural patterns that are preserved during the mutation process, and eventually rebuild phylogenic trees.

A secondary structure can be represented by an ordered labeled tree (see [15,2]). Thus, the secondary structure comparison, for which the pseudo-knots are not taken into account, can be reduced to a tree comparison.

For a few years, many ordered labeled tree comparison algorithms have been developed (see [12,4]). These algorithms were introduced by Selkow [9] and are based on the notion of edit distance or alignment score, which have since become a classical approach in the frame of sequence comparison (see [14]). Notice that these two notions which are equivalent in the case of sequence comparison, lead to two different problems in the case of tree comparison.

All the algorithms in this field are based on principles that are frequently used for sequence comparison, especially the global alignment [7] and the notion of edit distance or Levenshtein distance first introduced in [6].

This article mainly focus on the complexity analysis of the Zhang–Shasha algorithm [16], which is the reference when dealing with ordered labeled trees comparison, and will be divided in three parts.

We will firstly give some preliminaries concerning the ways to get different tree representations from an RNA secondary structure.

As a base for our analysis, we shall then present this algorithm using an original approach differing greatly with the Zhang–Shasha approach and allowing us to better emphasize the properties we shall use in order to analyze its complexity.

The third and main part of the article will deal with the exact complexity analysis of this algorithm, studying the distribution of trees according to the parameter "collapsed depth" defined on each node.

2. State of the art

2.1. Some definitions

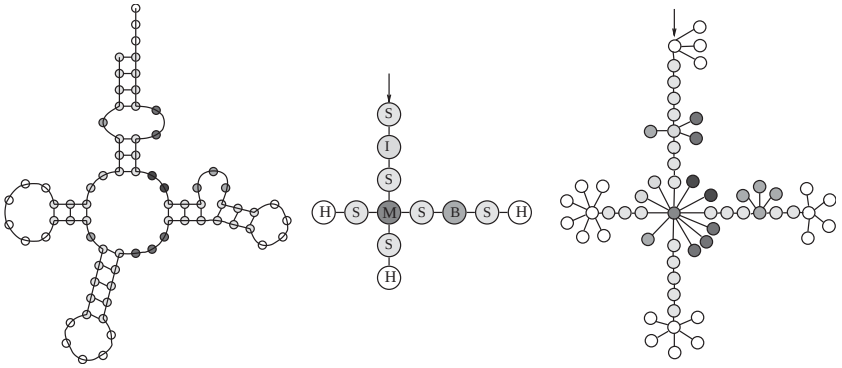
In the field of RNA secondary structure comparison, we are led back to the comparison of *ordered labeled trees*. A tree is said labeled if a label is associated to each node. A tree is said ordered if the edges incident to a given node are ordered cyclically. These are therefore trees for which the left-to-right order among the sibling nodes is significant (see [13] or [8]).

The RNA structure can hence be decomposed in a way where the labels of the nodes represent:

(1) either structural elements (sequences of paired bases, hairpins, loop, bulges, stems, …),
(2) or nucleic acids (A, C, G, U), Watson-Crick pairs, Wobble pairs, etc.

In that way, considering only the structural patterns, this tree representation can be rough (situation (1)) or refined (situation (2)). Below are represented:

- a RNA secondary structure on the left, the dots representing the nucleic acids,
- a relatively rough tree representation of this structure on the center where the following labels are used: a node labeled S(tem) represent a sequence of paired bases, a node labeled B a bulge, a node labeled I an inside loop, a node labeled M a multi-branch loop and a node labeled H an hairpin.
- a coding of the same structure on the right: "coding" in the way that the tree (with a good labeling of its nodes) makes it possible to come back to the secondary structure (see [13] or [8]).



To compare such trees, several methods exist, like the one that consists in presenting the tree structure like a parenthesis system. The tree comparison can this way be reduced to a measurement of the similarity between two strings (see [10]). We shall not extend much on such algorithms here. Other methods, based on *edit operations* exist, which also allow to measure the degree of similarity between two trees. Among these different methods, we can notably cite the computation of the *edit score between two trees* [16,11], but also the *computation of the smallest common super-tree* [3], and the *computation of the largest common sub-tree* [5]. These algorithms are extensions of the algorithm to compute the edit score between two sequences, applied on trees.

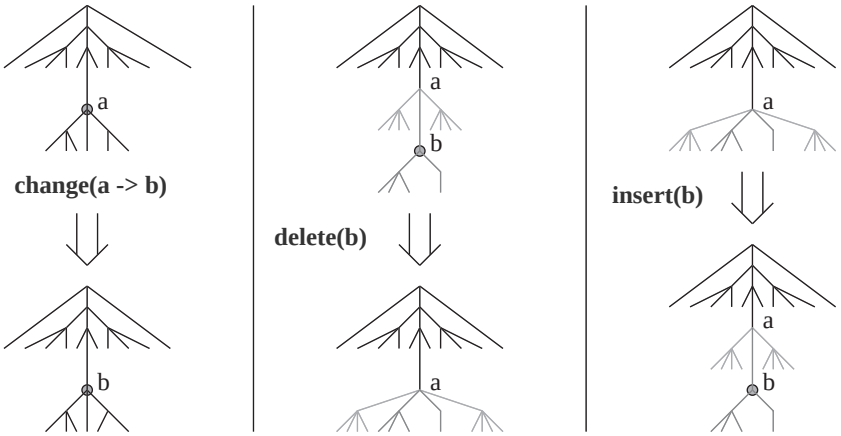
2.2. Edit score computation algorithm

Zhang and Shasha have developed an algorithm for the computation of the best edit score between two trees [16] that has a good time complexity. It is currently the reference in the field of edit score computation algorithm for two trees.

The goal of this algorithm lies in finding a sequence of edit operations to transform one tree into one another that maximizes the global score.

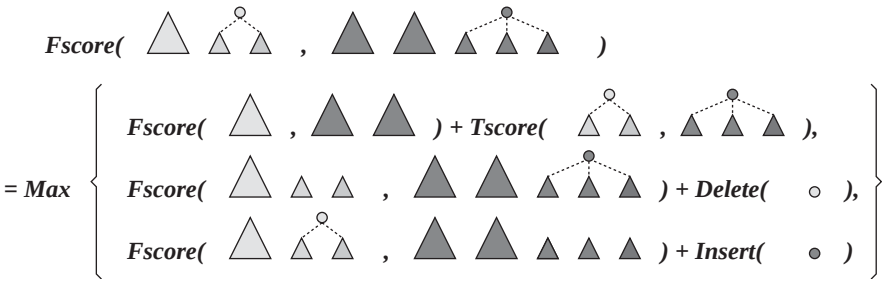
The edit operations are the only basic operations that are allowed to be processed on the nodes (and their labels) in order to transform a tree into one another. Three different types of operations exist: a label change of a given node or its conservation,

a deletion of a node or an insertion of a new labeled node. A score is associated to each of these operations.

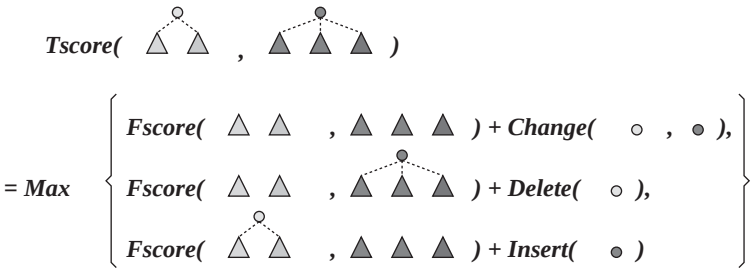


This algorithm, like most of the sequence comparison algorithms, can be implemented using the concept of dynamic programming.

The algorithm of Zhang and Shasha is based on the edit score computation between tree forests (a forest is an ordered sequence of trees). It lies on two main recurrences which help to determine the edit score between two forests and two trees.



We notice here that the recurrence that computes the edit score between two forests uses the edit score between two trees.



2.3. Scheduling of the computations

This algorithm lies on the concept of dynamic programming. Hence, it is necessary to schedule the computations in a way that all the values needed for a given computation are available when this computation happens.

The first step to undertake in order to analyze this algorithm is to understand why the scheduling of the computations that are processed using the suffix order on the nodes and a left branch decomposition of the tree are compatible with the recurrences given for *Tscore* and *Fscore*.

Definition. Let T be a tree. A special sub-tree of T is a sub-tree of T rooted at a node s of T where s is one of the starting node on the recursive factorization of T following the left branch decomposition.

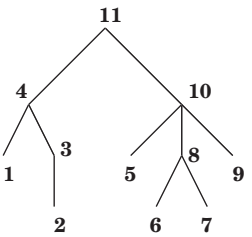
The roots of the special sub-trees of T can be obtained in the following way:

- first, delete the first leaf of T and all its ancestors: the last one (the root of T in this case) is the root of one of the special sub-trees of T,
- repeat this process on the resulting forest, starting at its first leaf, until the forest becomes empty.

Remark. Clearly, the numbers of special subtrees of T is equal to the number of leaves of T.

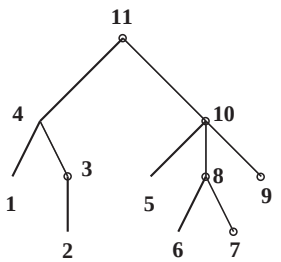
The scheduling of the computations for the recurrence formulas is based on:

- *A suffix order on the nodes of each tree*: The suffix order is obtained recursively, so that each node is preceded by its sons (see the following example),



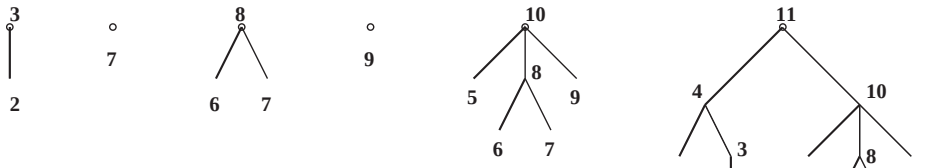
suffix order

- *A recursive pruning of the trees following their left-branch*: The starting nodes of the consecutive left branches give the roots of the special sub-trees of the tree (in the example, the starting nodes of the consecutive left branches are 11, 3, 10, 8, 7, 9),



decomposition following the left branches

- *A prefix ordering of the special sub-trees following the suffix order of their roots*: The following example shows the special sub-trees ordered following the suffix order of their roots, that is 3, 7, 8, 9, 10, 11, the last tree being the complete tree considered.



special sub-trees ordering

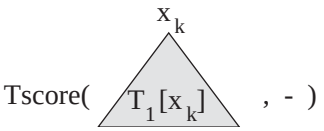
So, the edit score between two trees is obtained by computation of the recurrence formula for each pair of special sub-trees of the given trees using dynamic programming. Indeed, the two last sub-trees are the two given trees.

Now, let us try to understand why this scheduling of the computations allows that each term in the recurrence formula can be obtained in constant time from previously computed terms.

Given two trees T_1 and T_2, to obtain the edit score $Tscore(T_1, T_2)$, the computation of the recurrence formula is applied for each pair (S_1, S_2) where S_i $(i = 1, 2)$ is a special sub-tree of T_i, following the suffix order of their roots. So, the last computation that is processed is $Tscore(T_1, T_2)$.

Like that, each term in the recurrence formula is obtained in constant time from previously computed terms, as we shall see later on.

Let x_k be a one of the roots of the special sub-trees obtained through the left branch decomposition of T_1. Consider the step of the computation that is focused on this special sub-tree $T_1[x_k]$, that is:

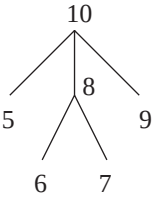


The node x_k is hence the greatest node of this tree relatively to the suffix ordering of the nodes of T_1.

In order to carry out this computation, the following computations are successively processed (by dynamic programming) with regard to the first tree: $Fscore_{(x_k)}(\emptyset, -)$, $Fscore_{(x_k)}(F_1[x_1..x_1], -)$, $Fscore_{(x_k)}(F_1[x_1..x_2], -), \ldots,$ $Fscore_{(x_k)}(F_1[x_1..x_k], -)$ where $x_1 < x_2 < \cdots < x_k$ are the nodes of $T_1[x_k]$ and $F_1[x_1..x_i]$ is the forest consisting in the sub-trees of $T_1[x_k]$ restricted to the nodes $x_1, x_2, \ldots, x_i$.

Thus $F_1[x_1..x_k] = T_1[x_k]$.

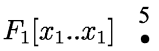
The same computation is also carried out with regard to the second tree.

Example. Given the tree $T_1[x_k] =$ 

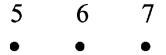
where $x_k = 10$.

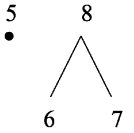
We have: $\{x_1, x_2, \ldots, x_k\} = \{5, 6, 7, 8, 9, 10\}$

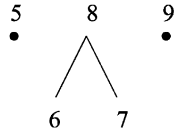
Then, the computations of *Fscore* are successively carried out on the following forests:

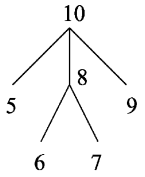
$\emptyset$ $\emptyset$ $F_1[x_1..x_1]$ 

$F_1[x_1..x_2]$ 

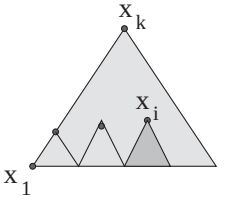
$F_1[x_1..x_3]$ 

$F_1[x_1..x_4]$ 

$F_1[x_1..x_5]$ 

$F_1[x_1..x_k]$ 

Now, the computation that determines $Fscore_{(x_k)}(F_1[x_1..x_i], -)$ where x_i is a node of $T_1[x_k]$ has to be processed.



Thus, $Fscore_{(x_k)}(F_1[x_1..x_i], -) = Fscore_{(x_k)}($ 

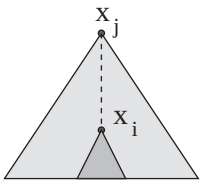
$, -)$.

The recurrence formula, in order to be valid, needs that the computation of $Fscore_{(x_k)}$ $(F_1[x_1..x_{i-1}],-) = Fscore_{(x_k)}($  $,-)$ has already occurred, which is indeed the case (preceding step). It also needs that the computation of $Tscore(T_1[x_i],-)$ has been previously carried out.

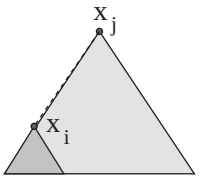
Two cases need to be considered:

(a) Either x_i is one of the roots obtained during the left branch recursive decomposition of the initial tree T_1 (x_i is the root of a special sub-tree). As $x_i < x_k$ and the computations $Tscore(T_1[x_i],-)$ are carried out according to the suffix order, $Tscore(T_1[x_i],-)$ was necessarily calculated during a previous stage of the computation.

(b) Or x_i is not one of the roots that were obtained during the decomposition. Then x_i belongs to a tree stemmed from x_j ($x_i < x_j \leqslant x_k$), node obtained during the left branch decomposition of T_1:



Yet, the decomposition is only carried out according to the left branch. So, the above picture is in fact



and hence, $Tscore(T_1[x_i],-)$ has already be obtained during the computation of $Fscore_{(x_j)}(F_1[x_{j_1}..x_i],-) = Tscore(T_1[x_i],-)$ where x_{j_1} is the smallest node (according to the suffix order) of $T_1[x_j]$.

So, with this scheduling of the computations, each term in the recurrence formulas is obtained in constant time from previously computed terms and hence, the algorithm is valid for the computation according to this decomposition process.

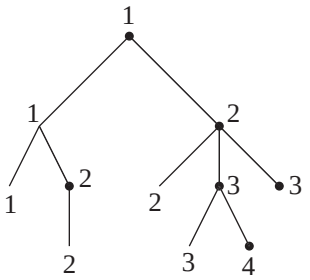
3. Complexity analysis of the Zhang–Shasha algorithm

Given two trees T_1 and T_2, the computation of *Tscore* is processed for each pair (S_1, S_2) where S_i $(i = 1,2)$ is a special sub-tree of T_i. Moreover, the computation of $Tscore(S_1, S_2)$ by dynamic programming using the recurrence formula, requires exactly $size(S_1).\ size(S_2)$ steps, each one of them in constant time. So, each node of the tree

T_1 (respectively T_2) takes place in the computations exactly the number of times it appears in a special sub-tree of T_1 (respectively T_2). This consideration leads us to define a parameter on trees called *collapsed depth*.

Definition. Let T be a tree and s a node of T. We define the collapsed depth $cd(s)$ of the node s as the number of special sub-trees of T which contain s.

The following example shows the collapsed depth of the nodes of the tree considered previously.



So, the time complexity of the Zhang–Shasha algorithm for the computation of the edit score between two trees T_1 and T_2 is

$$Complexity_{\text{Zhang}}(T_1, T_2) = CD(T_1).CD(T_2),$$

where

$$CD(T) = \sum_{x \text{ node of } T} cd(x).$$

3.1. An upper bound

Afterward, given a tree T, $|T|$ represents the size of T (number of nodes), $l(T)$ its number of leaves and $h(T)$ its height or depth (length of the longest branch).

Firstly, for a given node x of T, we have clearly

$$cd(x) \leqslant h(x) \leqslant h(T).$$

Therefore

$$CD(T) \leqslant h(T).|T|.$$

Secondly, the computation of *Tscore* is processed for every special sub-tree (there are $l(T)$ special sub-trees for a tree T), and, for each special sub-tree, requires a number of steps exactly equal to its size.

We deduce from that an upper bound for the complexity of the Zhang–Shasha algorithm.

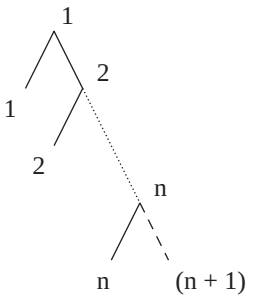
Proposition 1 (Zhang–Shasha [16]).

$$Complexity_{\text{Zhang}}(T_1, T_2) = \mathrm{O}(|T_1|.|T_2|.\min(h(T_1), l(T_1)).\min(h(T_2), l(T_2))).$$

3.2. Exact analysis

3.2.1. Complexity in the worst case

Clearly, the worst case situation happens when the trees look like the following (where the integers represent the collapsed depths of the nodes):



Hence, $CD(T) = \begin{cases} n(n+1) & \text{if } |T| = 2n, \\ (n+1)^2 & \text{if } |T| = 2n+1. \end{cases}$

So, the complexity of the algorithm in the worst case is on the order of $|T_1|^2.|T_2|^2$.

3.2.2. Average complexity

Let us consider the generating functions of trees according to their size and collapsed depth, that is:

$$f(q,t) = \sum_{n \geqslant 1, k \geqslant 1} a_{n,k} q^k t^n,$$

where $a_{n,k}$ is the number of trees having n nodes and a collapsed depth equal to k.

Hence, the average collapsed depth of trees having $n+1$ nodes is given by

$$CD(n+1) = \frac{1}{\mathscr{C}_n} \sum_k k.a_{n+1,k} = \frac{1}{\mathscr{C}_n} \left(\left[\frac{\mathrm{d}}{\mathrm{d}q} f(q,t) \right]_{q=1}, t^{n+1} \right),$$

where $\mathscr{C}_n = 1/(n+1)\binom{2n}{n}$ is the number of trees of size $n+1$, that is the nth Catalan number (see [1]).

Theorem 1. *The generating function of trees according to the size and collapsed depth verifies the functional equation*

$$f(q,t) = qt + qt.f(q,t)\frac{1}{1 - f(q,qt)}$$

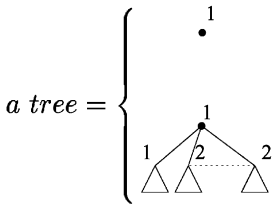
and

$$CD(n+1) = (n+1)\frac{\binom{2n-1}{n} + 4^{n-1}}{\binom{2n}{n}}.$$

Corollary 1. *The average complexity of the Zhang–Shasha tree edit algorithm for two trees T_1 and T_2 is on the order of*

$$|T_1|^{3/2}.|T_2|^{3/2}.$$

Proof. First of all, we consider the only way to factorize a tree into its root and an ordered list of sub-trees. The following figure shows this factorization with the collapsed depth of the roots and its sons.



It is easy to see that the collapsed depth of every node belonging to a sub-tree stemming from the root (with the exception of the first sub-tree) is the collapsed depth calculated in this sub-tree increased by one unit.

So, in the expression of the generating function $f(q,t)$, the term $qt.f(q,t)$ corresponds to the weight of the root and its first sub-tree and $1/1-f(q,qt)$ to the weight of the list (possibly empty) of the other sub-trees with the collapsed depth of each node increased by one ($q \to qt$).

Otherwise, a tree reduced to a single node contributes to qt in $f(q,t)$.
So,

$$f(q,t) = qt + qt.f(q,t)\frac{1}{1-f(q,qt)}. \tag{1}$$

If we put $q=1$ in Eq. (1), we obtain

$$f(1,t) = a(t) = t + t.a(t)\frac{1}{1-a(t)}.$$

Then $a(t)$ is the generating function for trees according to their size which is solution of the algebraic equation

$$a(t)^2 - a(t) + t = 0. \tag{2}$$

So,

$$a(t) = f(1,t) = \frac{1-\sqrt{1-4t}}{2} = \sum_{n\geqslant 0} \mathscr{C}_n t^{n+1}, \tag{3}$$

where $\mathscr{C}_n = 1/(n+1)\binom{2n}{n}$ is the nth Catalan number.

Let $F_Q(t) = [\frac{\mathrm{d}}{\mathrm{d}q} f(q,t)]_{q=1}$ and $F_T(t) = [\frac{\mathrm{d}}{\mathrm{d}t} f(q,t)]_{q=1}$.

As $f(q,t) = \sum_{n\geqslant 1, k\geqslant 1} a_{n,k} q^k t^n$, we have

$$\left[\frac{\mathrm{d}}{\mathrm{d}t} f(q,qt)\right]_{q=1} = F_T(t) \quad \text{and} \quad \left[\frac{\mathrm{d}}{\mathrm{d}q} f(q,qt)\right]_{q=1} = F_Q(t) + t.F_T(t).$$

By computing the derivatives of Eq. (1) with respect to q and t, respectively, we obtain, after some substitutions using Eq. (2)

$$F_Q(t) = t.a(t)\frac{1+a(t)^2 F_T(t)}{2t-a(t)},$$

$$F_T(t) = \frac{a(t)}{2t-a(t)}.$$

So, using expression (3) of $a(t)$, we obtain

$$F_Q(t) = \frac{t}{2}\,\frac{1-2t+\sqrt{1-4t}}{1-4t}.$$

The Taylor series expansion of $F_Q(t)$ gives

$$(F_Q(t), t^{n+1}) = \begin{cases} 4^{n-1} + \dbinom{2n-1}{n} & \text{if } n \geqslant 1, \\ 1 & \text{if } n = 0. \end{cases}$$

So, the first terms of $F_Q(t)$ are

$$F_Q(t) = t + 2t^2 + 7t^3 + 26t^4 + 99t^5 + 382t^6 + 1486t^7 + 5812t^8 + 22819t^9 \ldots$$

As $CD(n+1) = 1/\mathscr{C}_n(F_Q(t), t^{n+1})$, we have

$$CD(n+1) = (n+1)\frac{\dbinom{2n-1}{n} + 4^{n-1}}{\dbinom{2n}{n}}. \qquad \square$$

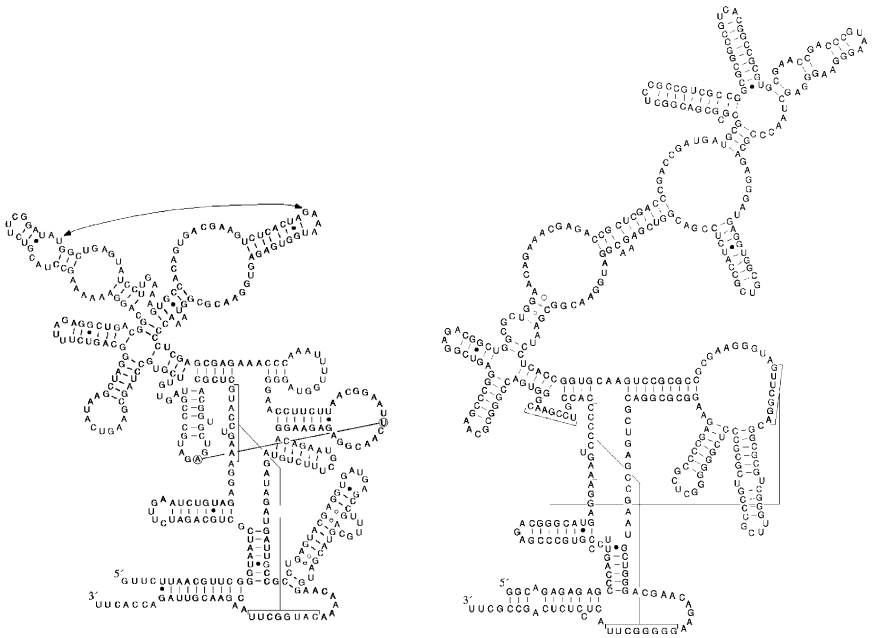
The corollary is an immediate consequence of this last expression using the fact that

$$\binom{2n}{n} = \frac{4^n}{\sqrt{\Pi.n}}\left(1 + \mathrm{O}\left(\frac{1}{n}\right)\right).$$

4. Biological considerations

All this work previously done helped us to have a better comprehension of the tree edit algorithm, and what are the parameters that constraint the complexity. Thus, we have been able to bring some modifications to this tree comparison algorithm, without changing its time complexity. These modifications were done in order to integrate some of the constraints of biological nature that are taken into consideration in the case of sequence comparison. The problem we have looked into concerns the merging of common factors (sub-trees in the case we consider) with the introduction of weights allowing us to penalize vertical gaps (along a branch of the tree) and horizontal gaps (between the sons of a given node). These modifications lead to a new algorithm that is not only more flexible than the original one, but also more adequate to biological problem. All these improvements and the new algorithm will be presented in a forthcoming paper. These algorithmic modifications are currently being validated on RNase P RNA secondary structures of prokaryotes.

An early example of RNase P RNA secondary structure comparison between Bacillus Subtilis and Haloferax Volcanii is given here. It shows the alignment obtained with a "vertical" and "horizontal" merging of the aligned regions. We can establish on this example that our algorithmic improvements avoid the dispersion of paired bases, thus merging the conserved areas (the aligned parts appear in darker characters).



Edition of B.subtilis and H.volcanii with merging

References

[1] N. Dershowitz, S. Zaks, Enumeration of ordered trees, Discrete Math. 31 (1980) 9–28.
[2] D. Gusfield, Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.
[3] T. Jiang, L. Wang, K. Zhang, Alignment of trees—an alternative to tree edit, in: CPM '94, 1994, pp. 75–86.
[4] P. Klein, Computing the edit-distance between unrooted ordered trees, in: Lecture Notes in Computer Science ESA '98 (Venice), Springer, Berlin, 1998, pp. 91–102.
[5] P. Kilpelinen, H. Mannila, The tree inclusion problem, in: Proc. Internat. Joint Conf. on the Theory and Practice of Software Development, 1991, pp. 202–214.
[6] V.I. Levenshtein, Binary codes capable of correcting insertions and reversals, Sov. Phys. Dokl. 10 (1966) 707–710.
[7] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (1970) 443–453.
[8] W.R. Schmitt, M.S. Waterman, Linear trees and RNA secondary structures, Discrete Appl. Math. 51 (1994) 317–323.
[9] S.M. Selkow, The tree-to-tree editing problem, Inform. Process. Lett. 6 (6) (1977) 184–186.
[10] B.A. Shapiro, An algorithm for comparing multiple RNA secondary structures, Comput. Appl. Biosci. 4 (1988) 387–393.
[11] B. Shapiro, K. Zhang, Comparing multiple RNA secondary structures using tree comparisons, Comput. Appl. Biosci. 6 (4) (1990) 309–318.
[12] K.-C. Tai, The tree-to-tree correction problem, J. ACM 26 (3) (1979) 422–433.
[13] M. Vauchaussade, X. Viennot, Enumeration of RNA secondary structures by complexity, in: Lecture Notes in Biomathematics, Vol. 57, Springer, Berlin, 1985, pp. 360–365.
[14] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. Assoc. Comput. Mach. 21 (1) (1974) 168–173.
[15] M.S. Waterman, Introduction to Computational Biology: Maps, Sequences and Genomes, Chapman & Hall, London, 1995 [Chaps. 13 and 14].
[16] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM J. Comput. 18 (6) (1989) 1245–1262.